9.3 Tuple Relational Calculus: Query Language for Relational Databases

- non-procedural/declarative language
- uses first-order predicate logic to express queries
- equivalent to relational algebra

A TRC Query is expressed as

{ t1.A1, t2.A2, ..., tn.An | COND(t1, t2, ..., tn, tn+1, ..., tn+m) }

where

- ti, ..., tn+m are tuple variables,
- Ai is an attribute of the relation on which ti ranges.
- COND is a formula in predicate logic involving the tuple variables t1, ..., tn+m where t1, ..., tn are the ONLY FREE VARIABLES (the remaining are BOUND by quantifiers)

  The syntax for COND is defined as follows:

  ATOMIC FORMULAS:

  1. R(ti) is an atomic formula where R is a relation and ti is a tuple variable.
  2. ti.A op tj.B is an atomic formula where op is one of <, <=, =, <>, >, >=
  3. ti.A op c is an atomic formula where c is a constant
  4. c op ti.A is an atomic formula where c is a constant

  EACH of the above atomic formulas evaluate to TRUE/FALSE

  FORMULAS:

  1. Each atomic formula is a formula
  2. if F1 and F2 are formulas then so are (F1 and F2), (F1 or F2), not (F1)
  2. if F is a formula and t is a tuple variable then so are (Exists t)(F), (Forall t)(F)

Query Examples: (These are the queries from problem 7.18 of the
El-Masri/Navathe text).

(1) Get names of all employees in department 5 who work more than 10 hours/week on the ProductX project.

```
{ t.fname, t.minit, t.lname |
      employee(t) and
      (Exists w)(Exists p)(works_on(w) and project(p) and
                           t.ssn = w.essn and w.pno = p.pnumber and
                           w.hours >= 10 and p.pname = 'ProductX') }
```

(2) Get names of all employees who have a dependent with the same first name as themselves.

```
{ t.fname, t.minit, t.lname |
      employee(t) and
      (Exists d)(dependent(d) and t.ssn = d.essn and
                 t.fname = d.dependent_name) }
```

(3) Get the names of all employees who are directly supervised by Franklin Wong.

```
{ t.fname, t.minit, t.lname |
      employee(t) and
      (Exists e)(employee(e) and t.superssn = e.ssn and
                 e.fname = 'Franklin' and e.lname = 'Wong') }
```

(4) Get the names of all employees who work on every project.

```
{ t.fname, t.minit, t.lname |
      employee(t) and
      (Forall p)(project(p) -> (Exists w)(works_on(w) and
                                          w.essn = t.ssn and
                                          e.pno = p.pnumber)) }
```

(5) Get the names of employees who do not work on any project.

```
{ t.fname, t.minit, t.lname |
      employee(t) and
      not (Exists w)(works_on(w) and w.essn = t.ssn) }
```

(6) Get the names and addresses of employees who work for at least one project located in Houston but whose department does not have a location in Houston.

```
{ t.fname, t.minit, t.lname |
      employee(t) and
      (Exists w)(Exists p)(works_on(w) and project(p) and
                           t.ssn = w.essn and w.pno = p.pnumber and
                           p.plocation = 'Houston') and
      not (Exists d)(dept_locations(d) and t.dno = d.dnumber and
                     d.dlocation = 'Houston') }
```

(7) Get the names and addresses of employees who work for at least one project located in Houston or whose department does not have a location in Houston. (Note: this is a slight variation of the previous query with 'but' replaced by 'or').

```
{ t.fname, t.minit, t.lname |
      employee(t) and
      ((Exists w)(Exists p)(works_on(w) and project(p) and
                           t.ssn = w.essn and w.pno = p.pnumber and
                           p.plocation = 'Houston') or
      not (Exists d)(dept_locations(d) and t.dno = d.dnumber and
                     d.dlocation = 'Houston') ) }
```

(8) Get the last names of all department managers who have no dependents.

```
{ t.lname | employee(t) and
            (Exists d)(department(d) and t.ssn = d.mgrssn and
                       not (Exists p)(dependent(p) and
                                      t.ssn = p.essn)) }
```